(12)

University of Southern California

Automatic Documentation Generation: The interaction of Text and Examples

Vibhu O. Mittal and Cecile Paris
USC/Information Sciences Institute

March 1993
ISI-RR-93-333

INFORMATION SCIENCES INSTITUTE

310/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

Automatic Documentation Generation: The interaction of Text and Examples

Vibhu O. Mittal and Cecile Paris
USC/Information Sciences Institute

March 1993
ISI-RR-93-333

To appear in the Proceedings of the 13th International Joint Conference On Artificial Intelligence (IJCAI'93), Chambery, France

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1993	Research Report

4. TITLE AND SUBTITLE

Automatic Documentation Generation: The interaction of Text and Examples

5. FUNDING NUMBERS

NCC2-250
DABT63-91-C-0025

6. AUTHOR(S)

Vibhu O. Mittal and Cecile Paris

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

RR-333

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

NASA AMES
Moffett Field, CA
94035

ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

To appear in the Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Good documentation is critical for user acceptance of any system, and empirical studies have shown that examples can greatly increase effectiveness of system documentation. However, studies also show that badly integrated text and examples can be actually detrimental compared to using either text or examples alone. It is thus clear that in order to provide useful documentation automatically, a system must be capable of providing well-integrated examples to illustrate its points. Precious work on example generation has concentrated on the issue of retrieving or constructing examples. In this paper, we look at the integration of text and examples. We identify how text and examples co-constrain each other and show that a system must consider example generation as an integral part of the generation process. Finally, we present such a system, together with example.

14. SUBJECT TERMS

documentation, natural language generation examples

15. NUMBER OF PAGES

6

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- DOE - See authorities.
- NASA - See Handbook NHB 2200.2.
- NTIS - Leave blank.

Block 12b. Distribution Code.

- DOD - Leave blank.
- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- NASA - Leave blank.
- NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Securitv Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

To appear in the Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France.

Automatic Documentation Generation: The interaction of Text and Examples

Vibhu O. Mittal and Cécile L. Paris

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
USA

Department of Computer Science
University of Southern California
Los Angeles, CA 90089
USA

Abstract

Good documentation is critical for user acceptance of any system, and empirical studies have shown that examples can greatly increase effectiveness of system documentation. However, studies also show that badly integrated text and examples can be actually detrimental compared to using either text or examples alone. It is thus clear that in order to provide useful documentation automatically, a system must be capable of providing well-integrated examples to illustrate its points.

Previous work on example generation has concentrated on the issue of retrieving or constructing examples. In this paper, we look at the *integration* of text and examples. We identify how text and examples co-constrain each other and show that a system must consider example generation as an integral part of the generation process. Finally, we present such a system, together with an example.

1 Introduction

Good documentation is critical for user acceptance of any system, and, increasingly, it comes in both conventional manuals as well as on-line facilities. These are often based on hypertext or similar retrieval methods. Advances in areas such as knowledge-based systems, Natural Language Generation (NLG) and multi-media now make it possible to investigate the automatic generation of documentation from the underlying knowledge bases. This has several important benefits: it is easily accessible; it avoids frequent problems of inconsistency (as the information presented is obtained directly from the knowledge bases); and not the least, it can take the user, and the dialogue context into account.

Examples can greatly contribute to the effectiveness of documentation. Empirical studies have found that the inclusion of examples in documentation can increase user comprehension, e.g., [Reder *et al.*, 1986], sometimes by as much as 100% [Charney *et al.*, 1988]. In order to provide useful documentation automatically, then, a system must not only generate good descriptions, but it must also provide examples. This raises at least two issues: (1) finding appropriate examples and (2), making effective use of them. While the first issue has been investigated previously, e.g., [Rissland, 1980; Rissland *et al.*, 1984], the second one remains largely unstudied. This is the issue we are addressing in our effort to build a user-oriented documentation facility for the Explainable Expert System (EES) framework [Swartout *et al.*, 1992].

Building on work in psychology, education, computational generation of examples and NLG, we are studying how text and examples interact with each other. In this paper, we argue that example generation should be considered an integral part of the documentation generation process, as examples *must* be *integrated* within the surrounding text. Indeed, psychological evidence shows that badly integrated text and examples can be detrimental compared to using either text or examples alone, e.g., [Chandler and Sweller, 1991]. We present a text generation system that achieves this integration by planning a presentation (text and examples) taking into account relevant factors in the generation of these two rhetorical devices. Finally, we present a specific example from our system, in which documentation about a specific programming language is generated.

2 Previous work and Open Issues

There have been considerable efforts in analyzing the type of examples that may be useful to present to the user (e.g., [Michener, 1978]), as well as in deciding whether an example should be retrieved or constructed, and on how to retrieve the appropriate example, e.g., [Rissland and Ashley, 1986; Rissland, 1980; Suthers and Rissland, 1988]. Other work has focused on *when* to present examples [Woolf and McDonald, 1984]. However, none of these studies looked at generating both text and examples, and thus did not look at the issue of ensuring their integration.

In NLG, researchers have used examples as one of the rhetorical strategies used to produce texts, e.g., [McKeown, 1985; Moore, 1989], but did not address the issue of choosing the examples to fit a specific description and integrating them into the text.

Finally, there is a large body of work on the use of examples in education, e.g., [Houtz *et al.*, 1973], cognitive science and psychology, e.g., [Reiser *et al.*, 1985], and documentation, e.g., [Charney *et al.*, 1988]. Although much of this work is not immediately computationally applicable, it suggests constraints to take into account when generating texts and examples.

Our work builds on the work mentioned above and our own analysis of documentation material to develop a system that generates appropriate examples in the context of a surrounding text to form a coherent, well-structured presentation.

The authors gratefully acknowledge support from NASA-Ames grant NCC 2-520 and DARPA contract DABT63-91-C-0025.

A LIST contains zero or more pieces of data elements. Examples of lists are:

```
(aardvark)              ;; a list of one element
(red yellow green)      ;; a list of several elements
(2 3 5 11 19)           ;; a list can contain numbers
(3 french fries)        ;; data elements can be of
                        ;; different types
```

Given the three lists: `(1 2)`, `(3 4)` and `(5 6)`, the following is also a list:

```
((1 2) (3 4) (5 6))     ;; a list can contain other lists
```

Figure 1: Using prompts to make explicit the information contained in the ordering of the examples [Touretzky, 1984].

3 Interactions between Text and Examples

In order to identify the relevant factors in the interaction between text and examples, the system must consider at least the following points:

1. what should be presented through text, examples or both?
2. will there be one or multiple examples? If more than one example, how many? what information should each one convey? How should they be ordered?
3. can the example cause additional text to be generated that may not otherwise have been presented?

To address these issues, example and text generation have to be considered together. For example, to decide which aspects of the presentation should be illustrated with examples, a system must know the important features to convey. But, feature importance depends upon the context, and is based on factors such as the type of text (e.g., tutorial vs reference) and what the discourse is about. Much of this information is important when constructing a text as well, and will already be available in the discourse planning context. A module for generating examples can thus take advantage of this information.

The interaction actually goes both ways: a decision on which aspects of the information are to be presented through examples affects the textual description as well.[1] Consider for instance, the following definition of the LISP concept `list`:

> A list consists of a left parenthesis, zero or more data elements, followed by a right parenthesis. *The data elements can be either symbols, numbers, or a combination of these two types.*

Compare this definition with that in Fig. 1. In the figure, the information that, above, is expressed in italics was instead communicated through examples.

Another source of interaction occurs when an example embodies more than one point, or when a group of examples illustrate a point together. In such a case, the system needs to generate a prompt (i.e., a marker focusing attention on the points being made; for example, the comments next to the examples in Fig. 1 are prompts). Since these prompts can be textual, they will also need to be planned by the text-planner. To plan them, the system needs to know not only what the examples illustrate but also what is implied by their ordering. Consider for instance, the definition in Fig. 1. The examples illustrate the fact that the data elements in a `list` can be of different types and in any number. The order of the examples is important as examples are introduced from simplest to most complex, each building on the previous one. In this case, the author chose to make explicit the information that is implicit in the ordering of the examples and included a comment (a prompt) for each example.

Furthermore, it might be necessary to include yet more text between the examples to ensure the coherence of the presentation, or in order to set up an example properly. This is also done in Fig. 1, where the three lists `(1 2)`, `(3 4)`, and `(5 6)` are introduced before the last example.

To generate coherent descriptions which include examples, the generation of examples must thus be tightly integrated within the generation process: text and examples must co-constrain each other. In the next sections, we describe our documentation context, present a system that achieves the desired integration and illustrate it with an actual scenario.

4 Generating integrated explanations

Our system is part of the documentation facility we are building for the Explainable Expert Systems (EES) framework [Swartout *et al.*, 1992], a framework for building expert systems capable of explaining their reasoning as well as their domain knowledge. In EES, a user specifies a domain model (in the high level knowledge representation language Loom [MacGregor, 1988][2]), as well as problem solving principles, i.e., methods for solving problems in the domain. Given these and a variabilized goal to achieve, EES generates an expert system to solve goals of the same form.

The problem solving methods have to be written in a specific plan language, INTEND. INTEND is specified in the Backus-Naur Form (BNF),[3] a fragment of which is shown in Fig. 2. The grammar contains productions, and, optionally 'filter-functions' on the productions, i.e., tests that have to be satisfied (for instance 'pred-relation-form-test' on the `pred-relation-form` production).[4]

The grammar of INTEND is quite complex, and thus provides a good test-bed for a documentation facility. With such an on-line facility, users can get information as to what might be wrong when a plan does not parse, as well as descriptions of the various constructs involved, together with examples.

The documentation for the grammar symbol `predicate-form`, whose BNF definition is shown in Fig. 2, is shown in Fig. 3. Consider the examples and the textual explanation in this figure. The first three examples are positive, while the fourth is a negative example (or counter-example). (All of the examples are from our domain of local area networks.) The mutual constraints of the text and the examples can be seen again in many places:

1. The examples illustrate features mentioned in the text, namely the syntax of the `predicate-relation-form`.

[1] Note that these issues are similar to the ones that arise in the planning and presentation of *other* explanatory devices – such as diagrams, pictures and analogies, e.g., [Feiner and McKeown, 1991].

[2] Loom is a KL-ONE type language.

[3] For use by the generation facility, the grammar is transformed into an equivalent form in Loom.

[4] Transforming the BNF form to a Loom representation was a fairly straightforward task. However, the filter-functions on the productions could not be extracted automatically. These were annotated by hand.

```
if-form := '( 'IF predicate-form 'THEN expression
        { 'ELSE expression } ') ;
restricted-expression := var-name  | concept-desc  |
            function-form  | predicate-form ;
predicate-form := pred-relation-form  |
        pred-logical-form  | pred-action-form ;
pred-relation-form :=
   '( relation-name restricted-expression + ')
     |> pred-relation-form-test ;
pred-action-form := action-form  |> pred-action-test ;
pred-logical-form :=
   '( 'AND predicate-form + ')  |
   '( 'OR  predicate-form + ')  |'( 'NOT predicate-form ');
function-form :=
   '( relation-name restricted-expression + ')
     |> function-relation-form-test ;
```

Figure 2: A fragment of the INTEND grammar.

2. The first three examples are introduced with 'background' text, to make sure they are understood as positive examples: "Examples of predicate-relation-forms are ... "
3. Because the positive examples are introduced with text, they occur together, rather than interspersed with the negative example.
4. The sentence "However, the following is not a ... " is generated to make a contrast between positive and negative examples.
5. The negative example selected causes the generation of additional text both *before* and *after* the presentation of the example. This is because the example is not just *not a* `predicate-relation-form`, but it is also a `function-form`, a different (but similar) construct which can be contrasted with the `predicate-relation-form`. Additional text is generated first to introduce the negative example as a contrast to the positive ones, and then to explain the differences between the two similar constructs.

This scenario also illustrates the other aspects that have to be taken into consideration when generating integrated text and examples:

1. It is not enough to know the important features to convey. The system also has to differentiate between *variable* features and *fixed* features. Fixed features are those that cannot vary. In this scenario, the fact that a `predicate-relation-form` must begin and end with a parenthesis is a fixed feature. On the other hand, variable features are those which can vary within a certain range in a positive example – in this case, the relation-name is a variable feature. It is usually necessary to provide several examples to communicate the variable nature of the feature [Clark, 1971]. In this case, several relation names are used in an attempt to ensure the user realizes their variable nature. On the other hand, it is not always necessary to explicitly state fixed features, as they will become obvious from examples.
2. The presentation order of the examples is especially important to communicate the *critical features* of a concept.

A predicate-form is a restricted-expression. It returns a boolean value, and the number of arguments in a predicate-form is equal to the arity of the relation. A predicate-form can be of three types: a predicate-relation-form, a predicate-action-form, or a predicate-logical-form.

A predicate-relation-form is a relation-name followed by some arguments. The arguments are restricted-expressions, such as variables, concepts, function-forms and predicate-forms. Examples of predicate-relation-forms are:

```
(INDICATOR-STATE LED-1        ON)
(HARDWARE-STATUS LANBRIDGE-2 FAULTY)
(CONNECTED-TO    DECSERVER-1 VAX-A)
```

However, the following example is not a predicate-relation-form, but a function-form, because the number of arguments is not equal to the arity of the relation:

```
(CONNECTED-TO DECSERVER-1)
```

The difference between a function-form and a predicate-relation-form is that the function-form takes one less argument than the arity of the relation, and returns the range of the relation, while the predicate-logical-form takes as many arguments as the arity and returns a boolean value.

A predicate-action-form is ...

Figure 3: The documentation for '`predicate-form`'.

Critical features are features which, if modified, cause the example to change from positive to negative. In this case, the relationship between the arity and the number of arguments is a critical feature. By contrasting the third and fourth example, which are identical except for the number of arguments, the pair highlights the critical feature. In general, examples should be pairwise maximally different if they are positive examples, and minimally different if they are a positive-negative pair [Feldman, 1972].

We now describe how our generation system can generate integrated explanations, and present a trace of the system as it generates the explanation presented in this scenario.

4.1 The Generation Framework

Our framework implements the integration of text and example within a text-generation system. More specifically, we use a text-planning system that constructs text by explicitly reasoning about the communicative goal to be achieved, as well as how goals relate to each other rhetorically to form a coherent text [Moore and Paris, 1989; Moore, 1989; Moore and Paris, 1992]. Given a top level communicative goal (such as `(KNOW-ABOUT HEARER (CONCEPT PREDICATE-FORM))`,[5] the system finds plans capable of achieving this goal. Plans typically post further sub-goals to be satisfied. These are expanded, and planning continues until primitive speech acts are achieved. The result of the planning process is a discourse tree, where the nodes represent goals at various levels of abstraction, with the root being the initial goal, and the leaves representing primitive real-

[5] See the references given above for details on the notation used to represent these goals.

```
( define-text-plan-operator
  :effect (know-about H (concept ?c))
  :constraints (and (isa? ?c object)
                    (isa? ?c ?parent))
  :nucleus (bel H (isa ?c ?parent))
  :satellites (((elaboration ?concept) *optional*)))

( define-text-plan-operator
  :effect (elaboration ?concept)
  :constraints (disjoint-covering ?c ?d-c)
  :nucleus ((setq ?d-j (order-maxim-of-end-weight ?d-c))
            (inform S H (disjoint-cover ?c ?d-j)))
  :satellites (((foreach ?d-j (know-about H (?d-j given ?c))))))
```

Figure 4: Some sample plans from our application.

ization statements, such as (INFORM ...) statements. The discourse tree also includes *coherence relations* [Mann and Thompson, 1987], which indicate how the various portions of text resulting from the discourse tree will be related rhetorically. This tree is then passed to a grammar interface which converts it into a set of inputs suitable for input to a grammar.

Plan operators can be seen as small schemas which describe how to achieve a goal; they are designed by studying natural language texts and transcripts. They include conditions for their applicability, which can refer to the system knowledge base, the user model, or the context (the current text plan tree and the dialogue history). Two of the plan operators used in our system are shown in Fig. 4. The first one is used to describe objects by describing a concept in terms of its parent and possibly elaborating on this description. The second one can be used to elaborate upon a description by describing the sub-types of a concept.

Using this framework, the generation of examples can be accomplished by explicitly posting the goals of providing examples while constructing the text, i.e., some of the plan operators include the generation of examples as one of their steps. This ensures that the examples embody specific information that either illustrates or complements the information in the accompanying textual description, and that the text to be generated will reflect the presence of examples. The constraints of the plan operators indicate how the text and the examples co-constrain each other. Because the same planning mechanism is used to plan the text and the examples, integration is achieved in a straightforward way.

4.2 A Trace of the System

We illustrate how our system integrates text and examples by working through the generation of the example shown in Fig. 3, that is, a description of a grammar concept for a non-expert user.

The system initially begins with the top-level goal: (KNOW-ABOUT H (CONCEPT PREDICATE-FORM)). The text planner searches for applicable plan operators, and, finding the first plan presented in Fig. 4,[6] posts the two subgoals indicated in the plan: one to give a make the hearer believe that (predicate-form) is a restricted-expression (its parent in the Loom hierarchy), and another (optional) to provide more information (elaborate).

[6]When several plans are available, the system chooses one using *selection heuristics* designed by [Moore, 1989].

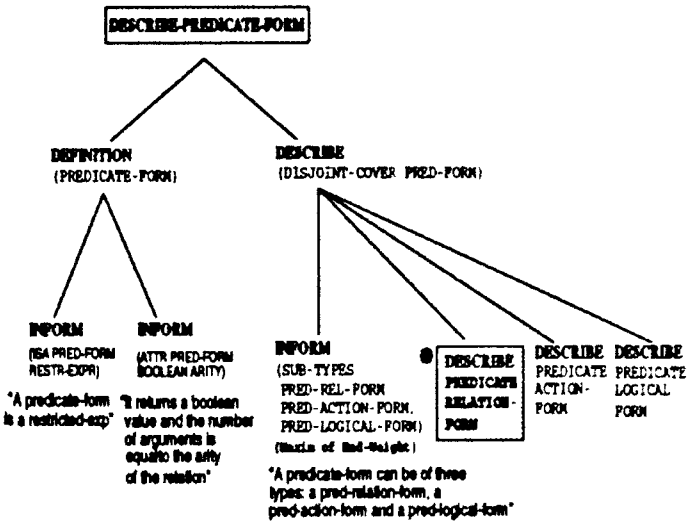

Figure 5: A skeletal fragment of the text plan generated for the initial text.

At this point, the discourse tree has two unexpanded nodes: (BEL H (ISA PREDICATE-FORM RESTRICTED-FORM)) and (ELABORATION PREDICATE-FORM). The planner expands the first subgoal by first indicating the concept-parent relationship ("a predicate form is a restricted expression") and then informing the user of the attributes differentiating a predicate-form from a restricted-expression ("it returns a boolean value ... ").

The goal to elaborate upon a concept is expanded in turn. The plan chosen here is the one shown in Fig. 4, which presents the different types of predicate-forms that exist, namely predicate-relation-form, predicate-action-form, and predicate-logical-form. Because these sub-types might be of differing complexity, and it is important to present the information from the simplest one to the most complex one (according to the maxim of end-weight in linguistics [Werth, 1984]), one step of the plan operator explicitly orders the sub-types before presenting them to the user. This ordering is done based on some general complexity heuristics: in this case, predicate-action-form is considered more complex than predicate-relation-form because it is allowed to have an action-form (a goal-posting construct) as one of its arguments. The predicate-logical-form is considered most complex because it is recursive in definition. After informing the user of the sub-types, goals to elaborate upon each of the sub-types are posted and expanded in turn. Each such elaboration results in posting the goal of describing each sub-type. This portion of the planning process is recorded in the skeletal text-plan shown in Fig. 5.[7]

Let's consider the expansion of the goal: (KNOW-ABOUT H (CONCEPT PREDICATE-RELATION-FORM)). Unlike for the

[7]All the text plans shown in this paper are simplified versions of the actual plans generated: they do not show the coherence relations that hold between the text spans resulting from this planning, and the communicative goals are not written in their formal notation, i terms of the hearer's mental states, for readability's sake.

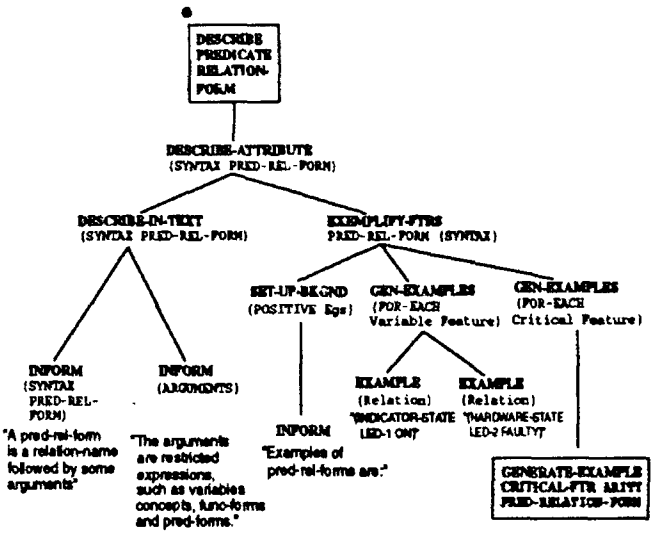

Figure 6: Plan fragment for the predicate-relation-form.

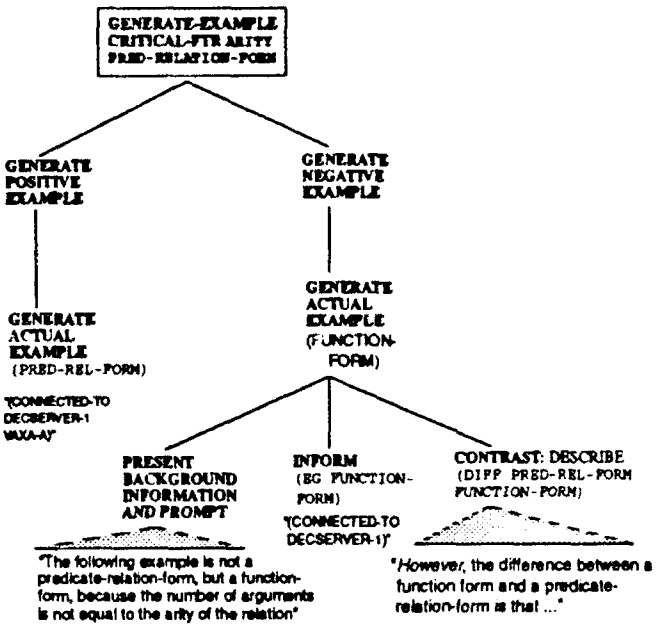

Figure 7: Text-plan fragment for the generation of the examples for the critical feature.

similar initial goal given to the system, the system does not pick the operator that defines the concept in terms of its parent, since the concept-parent relationship between a `predicate-relation-form` and a `predicate-form` has already been mentioned (since `predicate-relation-form` was introduced as a sub-type of `predicate-form`). Instead, a plan operator that describes the concept syntax is chosen. In this case, the syntax is: (`relation-name` `arguments`$^+$).

Instantiating the plan, the system realizes that it can describe the syntax by both text or examples (from the syntax definition, the system attempts to construct examples that match this syntax). The steps of the plan operator now compute the parameters that determine what get explained via text, via examples or both. In this case, the system determines[8] that there is one critical feature, i.e., the number of arguments in the `predicate-relation-form` must be equal to the arity of the relation, and one variable feature, i.e., the `relation-name`.

At this point, the system also determines that the parentheses do not need to be mentioned in the text as they will be mentioned in all the examples, are *fixed-features*, and more than one example will be presented. The system now has enough information to continue with the presentation planning process: it constructs the text: "a predicate-relation-form is a relation-name followed by ... ", and posts a goal to generate examples. The plan chosen to achieve this goal posts the goal to generate text to introduce the examples for the variable features as background, the goal to generate the examples for the variable feature, and the goal to generate examples for the critical features.

The system picks two positive examples to illustrate the variable feature. In general, as mentioned before, adjacent positive examples must be as different from one another as possible. In this case, the system picks two different relations, and the first two examples are generated. This part of the text plan is shown in Fig. 6.

Now the system must generate an example for the critical feature, i.e., that the number of arguments of a `predicate-relation-form` must be equal to the arity of the relation. Because this is a critical feature, the system decides to generate one pair of positive-negative examples to highlight the feature. The positive example of the pair will be presented first because the discourse tree shows that positive examples were already given and introduced with text. It is thus possible to simply give a new positive example without introducing it. To present the negative example, the system posts the goal of *contrasting* the positive example given.

There are two ways in which a negative example can be constructed in this case: by changing the number of arguments from two to three, or from two to one. The system constructs descriptions of both types, and checks to see whether one of these is a known concept (this can done easily using the classifier in Loom). It finds that a relation with one argument is a construct in the grammar, namely a `function-form`. It thus uses that as a negative example, after posting the appropriate goal to generate text to introduce the example. It also posts a goal to elaborate with text on the differences between a `function-form` and a `predicate-relation-form` to ensure that the user is not further confused by the use of a (possibly) new term. The relevant portions of the text-plan are shown in Fig. 7.

The planner continues expanding goals in this fashion, until all the goals are primitive speech-acts (such as `INFORM`). Finally, the completed discourse tree is passed to an interface which converts the `INFORM` goals into the appropriate input for the sentence generator. The interface chooses the appropriate lexical and syntactic constructs to form the individual sentences and connects them appropriately, using the rhetorical information from the discourse tree. For example,

[8]The system determines critical features and variable features by modifying the definitions and seeing whether an example of the modified definition becomes a negative example of the concept, using the Loom classifier.

it chooses "However" to reflect the CONTRAST relation.

5 Conclusions and Future Work

In this paper, we have shown how examples and text interact with and co-constrain each other. It is important to recognize this interaction in order to provide an appropriate, well-structured and coherent presentation to the user. We have also argued that a generation system capable of providing examples as part of its presentation must consider example generation as an integral part of the generation process. We have presented our documentation system which illustrates this idea.

While our system is already capable of generating integrated text and examples, some issues remain to be studied. In particular, we want to investigate issues related to goal (and example) interaction at a more global level than currently done: that is, how can a system "re-use" an example that was given to illustrate a different concept in a previous part of the dialogue.

References

[Chandler and Sweller, 1991] Paul Chandler and John Sweller. Cognitive Load Theory and the Format of Instruction. *Cognition and Instruction*, 8(4):292–332, 1991.

[Charney *et al.*, 1988] Davida H. Charney, Lynne M. Reder, and Gail W. Wells. Studies of Elaboration in Instructional Texts. In S. Doheny-Farina, ed., *Effective Documentation: What we have learned from Research*, chapter 3, pages 48–72. The MIT Press, Cambridge, MA., 1988.

[Clark, 1971] D. C. Clark. Teaching Concepts in the Classroom: A Set of Prescriptions derived from Experimental Research. *Journal of Educational Psychology Monograph*, 62:253–278, 1971.

[Feiner and McKeown, 1991] Steven K. Feiner and Kathleen R. McKeown. Automating the generation of coordinated multi-media explanations. *Computer*, 24(10):33–42, October 1991.

[Feldman, 1972] Katherine Voerwerk Feldman. The effects of the number of positive and negative instances, concept definitions, and emphasis of relevant attributes on the attainment of mathematical concepts. In *Proceedings of the Annual Meeting of the American Educational Research Association*, Chicago, IL, 1972.

[Houtz *et al.*, 1973] John C. Houtz, J. William Moore, and J. Kent Davis. Effects of Different Types of Positive and Negative Examples in Learning "non-dimensioned" Concepts. *Journal of Educational Psychology*, 64(2):206–211, 1973.

[MacGregor, 1988] Robert MacGregor. A Deductive Pattern Matcher. In *Proceedings of AAAI-88*, St Paul, MN, August 1988.

[Mann and Thompson, 1987] William Mann and Sandra Thompson. Rhetorical Structure Theory: a Theory of Text Organization. In L. Polanyi, ed., *The Structure of Discourse*. Ablex Publishing Co., Norwood, NJ, 1987.

[McKeown, 1985] Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, Cambridge, England, 1985.

[Michener, 1978] Edwina Rissland Michener. Understanding Understanding Mathematics. *Cognitive Science Journal*, 2(4):361–383, 1978.

[Mittal, 1993 forthcoming] Vibhu O. Mittal. *Generating descriptions with integrated text and examples*. PhD thesis, University of Southern California, Los Angeles, CA, 1993.

[Moore and Paris, 1989] Johanna D. Moore and Cécile L. Paris. Planning text for advisory dialogues. In *Proceedings of ACL 89*, pages 203 – 211, Vancouver, B.C., June 1989.

[Moore and Paris, 1992] Johanna D. Moore and Cécile L. Paris. Planning text for advisory dialogues: Capturing intentional, rhetorical and attentional information. Technical Report 92-22, University of Pittsburgh, Computer Science Department, Pittsburgh, PA, 1992.

[Moore, 1989] Johanna D. Moore. *A Reactive Approach to Explanation in Expert and Advice-Giving Systems*. PhD thesis, University of California – Los Angeles, 1989.

[Reder *et al.*, 1986] Lynne M. Reder, Davida H. Charney, and Kim I. Morgan. The Role of Elaborations in learning a skill from an Instructional Text. *Memory and Cognition*, 14(1):64–78, 1986.

[Reiser *et al.*, 1985] Brian J. Reiser, John R. Anderson, and Robert G. Farrell. Dynamic Student Modelling in an Intelligent Tutor for Lisp Programming. In *Proceedings of IJCAI 85*, pages 8–14. (Los Angeles), 1985.

[Rissland and Ashley, 1986] Edwina L. Rissland and Kevin D. Ashley. Hypotheticals as Heuristic Device. In *Proceedings of AAAI 86*, pages 289–297. 1986.

[Rissland *et al.*, 1984] Edwina L. Rissland, Eduardo M. Valcarce, and Kevin D. Ashley. Explaining and Arguing with Examples. In *Proceedings of AAAI 84*, pages 288–294. 1984.

[Rissland, 1980] Edwina L. Rissland. Example Generation. In *Proceedings of the Third National Conference of the Candian Society for Computational Studies of Intelligence*, pages 280–288. CIPS, Toronto, Ontario, May 1980.

[Suthers and Rissland, 1988] Daniel D. Suthers and Edwina L. Rissland. Constraint Manipulation for Example Generation. COINS Technical Report 88-71, Computer and Information Science, University of Massachusetts, Amherst, MA., 1988.

[Swartout *et al.*, 1992] William R. Swartout, Cecile L. Paris, and Johanna D. Moore. Design for explainable expert systems. *IEEE Expert*, 6(3):58–64, 1992.

[Touretzky, 1984] David S. Touretzky. *LISP: A Gentle Introduction to Symbolic Computation*. Harper & Row Publishers, New York, 1984.

[Werth, 1984] Paul Werth. *Focus, Coherence and Emphasis*. Croom Helm, London, England, 1984.

[Woolf and McDonald, 1984] Beverly Woolf and David D. McDonald. Context-Dependent Transitions in Tutoring Discourse. In *Proceedings of AAAI 84*, pages 355–361. 1984.